FILE PROCESSING EFFICIENCY ON THE CONTENT ADDRESSABLE FILE STORE

C.H.C. Leung and K.S. Wong

Department of Computer Science
University College London, Gower Street, London WC1E 6BT, UK

ABSTRACT

The file processing efficiency of the ICL Search Engine CAFS (Content Addressable File Store) is studied by means of detailed simulation experiments. The suitability of CAFS to different file structures and processing operations is critically assessed, and processing environments have been identified where the use of CAFS offers only marginal benefits and where it is actually counter-productive. We also provide concrete recommendations concerning the optimal deployment of CAFS as well as guidelines for the tuning of its performance parameters. In addition, the strengths and weaknesses of the CAFS architecture are assessed and some improvements to its current design are proposed.

1. INTRODUCTION

The principal aim of this paper is to study the file processing efficiency of the ICL backend database machine, the Content Addressable File Store (CAFS), and quantify the extent of its performance advantage over conventional direct access storage devices (DASD). This study also identifies the processing environments in which CAFS is best suited and those in which it only offers marginal improvement, as well as critically assess the type of design philosophy adopted in the construction of CAFS.

The objective of this work is four-fold. Firstly, while there is no doubt that CAFS will offer some performance improvement over conventional DASD, it is a costly resource and if it fails to confer sufficiently impressive benefits in a given processing environment, then it may not be worthwhile to install it at all. We shall therefore quantify its precise extent of improvement for different types of file sizes and processing operations.

Secondly, even if CAFS is available in an installation, it may not be wise to use it for every type of processing indiscriminately because while it may be highly efficient for certain operations such as fuzzy matching and multiple key search involving complex selection expressions, it may not be well-suited to others; by freeing CAFS from operations to which it is unsuited, it could be effectively deployed to satisfy competing demands for which its potential could be exploited to the full. It is also the aim of this study to categorise the common file processing operations into appropriate types so as to guide decisions as to when CAFS ought to and ought not to be invoked.

Thirdly, in CAFS processing, there exists a number of parameters, such as cell size and the degree of amplification in a drive which may be fine tuned to give maximum performance benefits. It is also the aim of this study to discover how these parameters affect processing efficiency and how they may be adjusted to optimise performance.

Finally, as an engineered product, CAFS may suffer from possible design drawbacks. Part of the aim of this study is to critically examine the CAFS architecture and identify it strengths and weaknesses; this will help to advance the design of future database machines using similar philosophy.

Not much previous work appear to have been done on the performance evaluation of CAFS. In [7], useful attempts to quantify the performance advantages of CAFS are made but are chiefly confined to savings in terms of

processing power and the extent to which CAFS relieves the mainframe of CPU cycles in controlling file operations. The overall performance gains in terms of speed and response time is not considered there. In this study, we shall assess the overall efficiency of CAFS in the processing of common file organisations by a series of simulations experiments which take into account both the ordercode instruction execution as well as the disc access overheads. Altogether, well over 100 separate simulation experiments have been performed. The chief contribution of this work is the detailed performance assessment of a production database machine system not undertaken before. In addition, we are able to condense our experimental findings into a simple set of usable performance figures for effectively quantifying the operational efficiency of CAFS for different file structures and processing operations, as well as to provide concrete recommendations to guide the efficient deployment of CAFS in practical processing situations.

2. PERFORMANCE CHARACTERISTICS OF CAFS

In this section and the next, we shall briefly describe the performance features and data structures of CAFS. Certain details are included insofar as they affect our construction of a meaningful simulation model. For a more complete description of the operation of CAFS, the reader is referred to [3,4].

Searching of CAFS files is initiated by software in the mainframe machine and is coordinated by the CAFS controller. There are special registers in the controller and prior to searching they are primed with the data values to be compared with fields in the file. The actual comparison is carried out on-the-fly by hardware that interprets the record format and matches data from the disc against the values in the registers. Simultaneously with the comparison process, data fields from the record are retrieved and stored for later transmission to the mainframe. At the end of each record, the results of the comparisons are evaluated by a special microprogram prepared in the mainframe and loaded into the controller. If the record is a hit, the retrieved data fields are preserved in the controller and sent to the mainframe, otherwise the buffer used for retrieval is emptied and comparison with the next record commences.

The CAFS controller is centred around a small computer which controls and coordinates a number of hardware sub-units. All transfers are buffered in this control processor, which is attached via a single channel to the mainframe. The principal sub-units of interest are: (a) Control Processor, (b) Direct Access Unit, (c) Associative Searching Unit, and (d) Record Retrieval Unit. The main functions of the Control Processor are task scheduling and resource management. The Direct Access Unit provides standard conventional direct access processing facilities. CAFS disc drives can be connected to a standard controller in addition to the CAFS controller and can be switched between the two, enabling the drive to be used in CAFS mode or conventional direct access mode.

The function of the Associative Searching Unit is to execute parallel searches on a multiplexed data stream obtained from the concurrent reading of several disc drives. In addition, each of the disc drives is usually equipped with read amplifiers so that up to ten heads on any one drive may read data in parallel. The Associate Searching Unit permits up to 16 key and mask registers together with corresponding comparators ($=$, $\neq$, $>$, $<$, $\geq$, $\leq$) to be allocated to any particular search task, allowing up to seven such tasks to run concurrently. After all key comparisons for a record have been made, a microprogram that has been specially compiled and loaded into it for the record selection expression in question will be invoked. This program combines the boolean results of each field comparison into a truth value which dictates whether the retrieval unit retains or discards the whole record. It also supports **quorum search** which allows weights to be attached to truth subexpressions and a threshold to be specified for the overall result so that any record which achieves a value exceeding this threshold will be retrieved. Before a search is started the Record Retrieval Unit is primed with a target list of fields for retrieval from hit records; this allows the contents of the designated fields in each hit record to be retained.

3. CAFS DATA STRUCTURES

The storage medium which supports CAFS implementation is the EDS60 disc which adopts a **count-data** format. The EDS60 has 406 cylinders, each of which is made up of 20 tracks. Each track is recorded as a series of 15 sectors, each sector being made up of a count block and a data block. These blocks are labelled from 0 to 14. The data on each track is considered by CAFS as a cyclic sequence so that block 0 logically follows on from block 14. Each record comprises a sequence of data fields followed by a 2-byte trailer field. The trailer field defines the end of each record and, during searching, triggers the process of combining the evaluations of its separate fields together to form a hit/no hit evaluation of the record as a whole. Rather than being identified by physical position within the

record, CAFS fields are self-identifying. The first byte of each field is used as an identifier byte. The field identifier and length are stored as a part of the field; the identifier indicates the type of data that exists in the field. CAFS fields are also variable in length, the second byte of each field giving the field's overall length in bytes. The remainder of the field is the actual user data. Successive trailer fields on a track must be at least 16 bytes apart to allow the hardware time to evaluate each record; 16 bytes is therefore the minimum record length. The retrieval unit's temporary buffer is 1024 bytes long but, as the data in the buffer is enveloped in control information, the full 1024 bytes are not available for retrieved fields. The number of control bytes required depends on the type of search that is being carried out and the practical maximum record length is taken as about 1000 bytes to allow the whole of every record to be retrieved. The area within the file to be searched for a given enquiry is specified to the controller as a list of search areas in the form: drive, cylinder, track, size. Each such search area is a cylinder or part cylinder and is called a **sucket**. The smallest sucket size is one whole track and each CAFS data record must lie entirely within one track.

The physical structure of a CAFS file can be termed a cellular serial one. The file extent is divided into a series of storage cells whose size may vary from one disc track to a cylinder depending on the particular requirements of the application. Although a sucket is defined by a dynamic search task and a cell is a static file portion, they frequently coincide in practice; in this study, we shall use these terms interchangeably. Any search task is then directed to one or more cells which are exhaustively scanned. In many applications a half-cylinder is used as a cell and then using ten head read the whole cell may be searched in one revolution of the disc pack. Since access to a record within a cell is associative, the physical location is irrelevant unless there is an applications requirement to maintain records in a given sequence. CAFS files can be totally unindexed so that the whole file will always be scanned for every search of that file. Alternatively, an index can be established for a CAFS file containing data ordered on some key, which is necessarily coarse and only resolve to the storage cell level.

4. EVALUATION OF PERFORMANCE

Here we are primarily concerned with quantifying the performance benefits of CAFS compared with conventional DASD for the three most common file organisations: sequential, indexed sequential, and hashed random. We shall consider the processing of these organisations via both their native access methods as well as other more flexible processing operations on them such as skip sequential processing, exhaustive search based on complex selection expressions, and binary search. Both the I/O and processor overheads will be taken into account in our evaluation. Although CAFS allows concurrent search tasks to be activated simultaneously, we shall in our experiments disregard any possible inter-task interference; this will help to eliminate any extraneous factor which will almost certainly cloud the central issues. For the same reason, although CAFS also permits the fragmentation of files into extents, the effect of which on performance is no doubt significant [6], we shall also disregard any such fragmentation and assume that all files are contiguously stored. In our experiments, unless otherwise indicated, the following are assumed: average values are computed from sample sizes of 500; the records are 100 bytes long; the blocking factor is 10; the drive amplification (i.e. maximum allowable number of active heads per drive) is 10; and the CAFS cell size is 10 tracks.

4.1 Evaluation of Sequential Organisation

4.1.1 Direct Processing

For flexible retrieval, it is frequently necessary to locate an arbitrary record from a file; this type of processing shall be referred to as direct processing in this study. Direct processing is often invoked by a search expression. The number of key terms in the expression is used to determine the number of ordercode instructions required to perform the key matching task. The relationship between the total number of instructions I and the number of selector keys K in processing a record is shown in [7] to be

$$I = K * (22 + 42m) ,$$

where m is the term degeneracy and is defined to be the average number of different values taken by a given data item in a record. The number of instructions when divided by the CPU power, usually expressed as the number of million instructions per second (mips), gives the CPU time to process the record. Therefore the time required to process a block of records can be estimated and then used to determine the latency penalty incurred in processing successive blocks. With conventional DASD, it is generally true that the processing of successive blocks would require more time than is allowed for by the inter-block gap so that in our model at least one rotation is assumed to be required for processing each block. With a large number of key terms and a high term

degeneracy, more than one rotation may be sometimes needed. These searches may be described as process bound, although they are only so relative to the processing power of the CPU; process bound searches are studied in greater detail in the next section.

The seek time of the EDS60 ranges from 10 milliseconds (ms) to 75 ms with an average of 35 ms. The seek characteristics of the EDS60 is nonlinear, and the seek time used in our model is that obtained from the actual head movement graph given in [4]. In context scanning where the hardware is normally able to start scanning at any alternate block, an average latency of one block at a time, or less than 1.7 ms is incurred; transfer time is about 1.7 ms per block. The associative search channels are required to be in step with logical records within the data blocks before starting their search, and since the logical records may cross block boundaries, it is generally necessary to re-read the first block at the end of the scan. Transfer time for searching a complete track is therefore normally 16 block times.

With CAFS implementation, the cell size specifies the number of tracks per sucket. Since the EDS60 is assumed to be equipped with 10 heads, the intuitive candidates for efficient cell sizes could only be either a factor or a multiple of 10, i.e. 1, 2, 4, 5, 10 or 20 tracks in order to avoid certain heads being unused in the scanning of a cell. During context scanning, all records in up to 10 tracks can be searched in one rotation. A 20-track cell still requires 2 rotations to be completely scanned, however; the effect of cell size on performance will be studied in greater detail in Sections 4.2 and 4.3.

There are two common ways in which a record in a sequentially organised file can be located: full search from the beginning of the file, and binary search. In our experiments, all such searches will be conducted using the latter algorithm as we found that it is always significantly faster than the former. With binary search [8], the highest (last) record in each cylinder is used to determine the next cylinder to be examined. The time to access this record is assumed to be 1 rotation due to the synchronisation of the index marker [5]. Once the required cylinder is located, the tracks are searched logarithmically in a similar manner, except this time, switching of read/write heads is done electronically and takes negligible time. Finally the located track is searched sequentially, block by block. The block on the track where the record is found is assumed to be random although strictly speaking it can be calculated from the record number. Such refinement is not necessary as it has little effect on the overall timing when averaged over large samples.

Fig. 1 shows the experimental ratios of the DASD processing time to that of CAFS for files of sizes (measured in number of records) 5000, 10000, 15000, 20000, 25000, and 30000. We see that this ratio does not seem to depend on the file size. It is possible to have a least square fit through these points, and we find that the resultant intercept is 2.52 with a standard error of 0.06. This indicates that the direct processing of a DASD sequential file is likely to take about two and a half times as long as one implemented in CAFS.

4.1.2 Exhaustive Sequential Search

For exhaustive sequential search, considerable benefit can be gained by using CAFS. Here the processing is similar to direct processing under full search except that the search continues to the end of the file. As remarked above, with conventional DASD, there is a significant difference in search speed between process bound and non-process bound operations. For example, with 16 key terms selection, a term degeneracy of 4, and a CPU power of 1 mips, the processing time of a block of records will exceed one rotation time of the EDS60 and the overall efficiency is greatly reduced: an experiment has been performed on a file of 32000 records with a blocking factor of 9, and we find that the mean processing time for 16 key terms selection is 192 seconds. Under identical conditions using 1 key term selection, the processing time is reduced by 46.9% to 102 seconds. However, since most present day computers are very powerful (often in excess of 5 mips), we shall in subsequent experiments be mainly concerned with non-process bound searches, although it must be acknowledged that process bound search could cause a substantial degradation in performance in conventional DASD. On the other hand, CAFS hardware can execute a search using up to 16 key terms in a complex selection expression with negligible difference in the elapsed time: under identical conditions, the above processing experiment when applied to CAFS yields a time of 1.9 seconds for both 1 and 16 key terms selection, giving speed improvements of 56 and 101 times respectively. We have intentionally left the performance of skip sequential processing for sequentially organised files out of consideration because the ICL record format is implemented in **count-data** rather than **count-key-data** format, and scanning keys at rotation speed is only possible with the latter format. However, skip sequential processing is considered in the context of indexed sequential files.

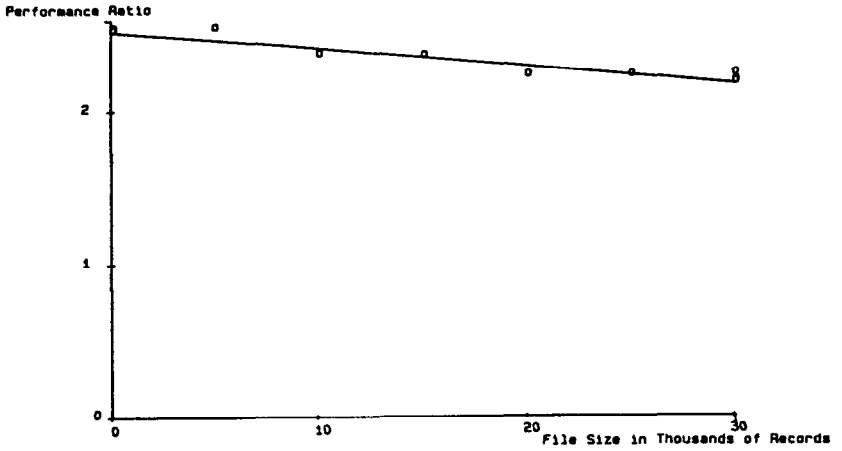

Fig. 1 Direct Processing Time Ratio: DASD/CAFS - Sequential Files

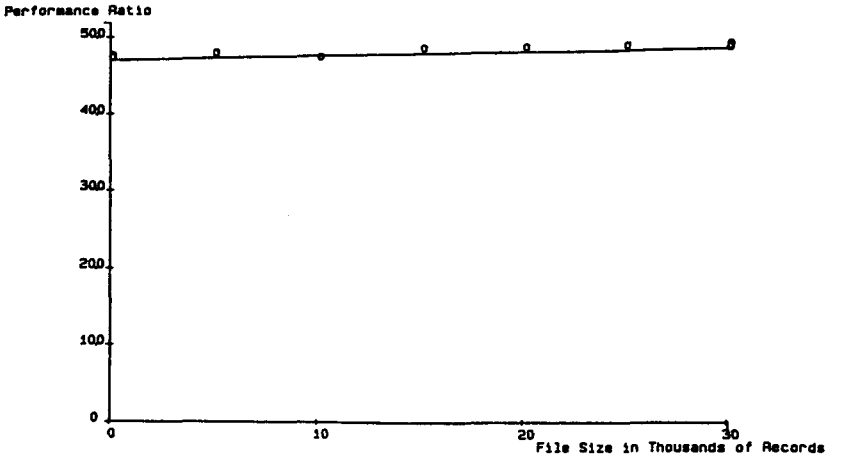

Fig. 2 Exhaustive Search Time Ratio: DASD/CAFS - Sequential Files

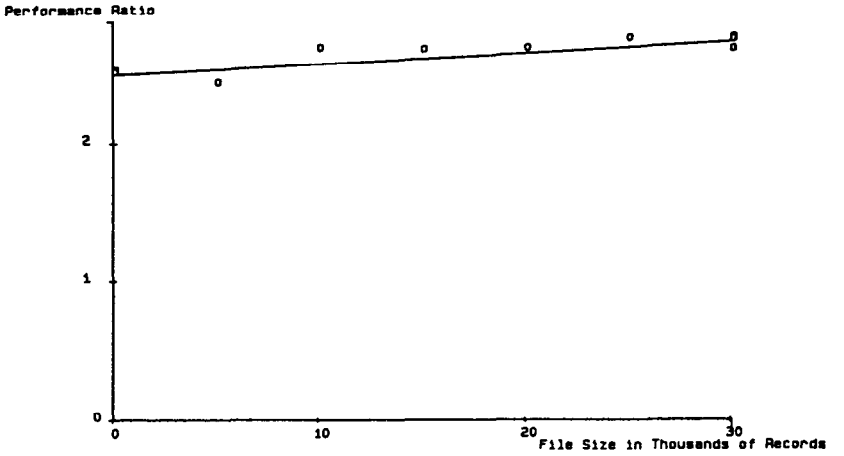

Fig. 3 Direct Processing Time Ratio: DASD/CAFS - Indexed Sequential Files

With exhaustive searching of sequential files, Fig. 2 displays the DASD to CAFS processing time ratios obtained for file sizes ranging from 5000 records to 30000 records as for Fig. 1. The intercept of the least square line through these points is found to be 47.0, with a standard error of 0.4. This is a much more impressive 18-fold improvement over the direct processing case.

4.2 Evaluation of Indexed Sequential Organisation

4.2.1 Direct Processing

In CAFS indexed sequential files, the sequential index, as pointed out earlier, is fairly coarse and is only resolved to a storage cell level, and is accordingly small in size. The mainframe software that handles searching uses the sequential index whenever a value for a data item declared as associated with the sequential index is encountered, thus automatically selecting only the relevant suckets. During the processing of an indexed sequential file, the sequential index is usually stored in mainframe memory. Searching of index is done in mainframe and the time taken is assumed to be negligible. Under conventional implementation, it is rarely true that the entire index could be accommodated in main memory as it is often much denser than the corresponding CAFS index, and consequently occupies much more space with the result that searching through it will have to be done externally.

First we shall consider processing performance in conventional DASD in our model. For the ICL indexed sequential organisation, the file is initially loaded to a preset percentage which by default is 90%. The distributed free space is used for new additions. There is a cylinder overflow area for each data cylinder. In our model, the number of tracks in each overflow area here is fixed to one. Initially each block is only partially filled. The blocking factor together with the packing density determines the total number of prime data blocks in the file. Depending on the file size, a multilevel index may be generated. The size of this index is determined by the number of prime data blocks, the size of index records and the blocking factor of index records. The index record size here is taken as 12 bytes to include an 8-byte key, 1 control byte and 3-byte block pointer. The index blocking factor is fixed at 10. For each of the records to be processed, a search through the index levels will locate the pointer to the prime data block. Since overflow blocks are chained to the prime data block and treated as its continuation, overflow records can be located by searching through this chain. The time to locate a record can therefore be divided into searching the index and searching the data blocks.

Searching through an index requires, first of all, a random seek to the highest index level. Subsequent seeks are one track seeks. The number of seeks is the smaller of the index levels and the number of cylinders occupied by the whole index. Each index block has to be transferred to main memory, deblocked and processed. We adopt the somewhat simplifying assumption that, on average, that only half of the records in each index block have to be processed. The timing considerations therefore include half a rotation to locate the index block, the block transfer time, and the processing time of half of the records. The number of key terms to decide a hit index record is obviously one. Searching the data blocks begins at the prime data block. It requires a random seek. If the record is in this block, on average half the records have to be searched. If the record is not in this block but in an overflow block, then all the records in this block have to be checked to arrive at this conclusion. The time taken to process a data block is similar to that of an index block. As mentioned earlier, overflow records (blocks) are stored in the same cylinder, so that there is no seek delay. An overflow rate (1%) is used to determine whether a search to the overflow chain is necessary.

With CAFS implementation, the default packing density is again 90%. In our model, we assume that insertion is made into its logical position in the cell and the record with the largest key is overflowed into a separate overflow area. We also suppose that there is a separate overflow area (the size is assumed to be at most 10 tracks) for each prime data cell. Within each overflow area, records are in key sequence. All overflow areas are separate from the prime data cells so that accessing an overflow area requires an extra seek, but all records within the overflow area can be accessed together. The overflow information is kept in an overflow vector in an index table and is available to the mainframe software when the file is being processed. Therefore, in direct processing, there is no difference in accessing records in prime area and overflow area. Each directly accessed record takes a random seek, one block rotation delay and time to scan a cell; and since scanning can stop as soon as record is found, the last timing factor is also random. The empirical performance ratios of DASD to CAFS is shown in Fig. 3; the least square line intercept, which indicates the average improvement, is found to be 2.51, with a standard error of 0.08. In addition, experiments were performed on a file of 32000 records in which the DASD index search times are ignored, and we find that the time taken to

access a record in CAFS takes an average of 57.5 ms, while it takes an average of 59.5 ms in conventional DASD, each average being computed from 1000 samples - this performance difference is hardly noticeable.

Experiments have also been performed to gauge the effect of the cell size on performance, using a file of 32000 records and cell sizes of 2, 4, 5, 10, 20. The respective processing times are observed to be 57.3 ms, 57.7 ms, 57.5 ms, 57.3 ms, and 72.3 ms. These observations suggest that there is a pronounced degradation in performance by using a 20 track cell size. A plausible explanation for this being that, if the overflow chains are not excessive, then generally only a single cell needs to be searched. Since any cell size not exceeding 10 tracks can be scanned in one revolution, there is no performance difference in obtaining a record for these cell sizes. However, for cell sizes exceeding 10 tracks, then more than 1 revolution may be necessary to locate the record since the record may be found in 11th or higher tracks of the cell, hence giving rise to the observed degradation in performance.

4.2.2 Skip Sequential Processing

One of the key advantages of indexed sequential files is the option of skip sequential processing. When the **hit rate** - i.e. the percentage of records that have to be processed - is low, records, whole tracks and even cylinders may be skipped. Although a sequential file that uses a record storage format in which keys are separate from data can also skip records, every key nevertheless has to be checked. An indexed file can allow more marked savings to be made as only the index entries need to be read, and these show which tracks and cylinders can be skipped. A high blocking factor generally has an undesirable effect for skip sequential processing because as more records are brought together to form a block, it becomes more likely that a large block of records will have to be accessed in order to process a single record only. We suppose that the block hit rate B for a block of n records is related to the record hit rate p by:

$$B = 1-(1-p)^n,$$

which for independent record activities appears to be a reasonable assumption. If the record hit rate or the block hit rate is 100% then this will reduce to sequential processing of the entire file. If the block hit rate is less than 100% skip sequential processing (of prime blocks) is applied. Because overflow blocks are not usually full it is difficult to estimate the activity of these blocks. For simplicity, the number of overflow blocks to be processed is taken as a certain percentage (1%) of the total prime blocks. Each overflow block processing requires half a rotation delay, block transfer time and CPU time.

With CAFS implementation, the record hit rate is used to determine the cell hit rate in a similar manner, and non-hit cells can be skipped. When a cell is a hit, the whole cell is scanned as it usually contains a large number of records which implies that there will be more hit records in it compared to a single block and they are likely to be scattered throughout the cell. The amount of overflow areas to be searched is also taken as a certain percentage (1%) of the total storage cells in the file.

Fig. 4 and Fig. 5 show the sensitivity of the processing time to the hit rates (1%, 5%, 10%, 25%, 50%, 75%, and 100%) for both DASD and CAFS for a file of 30000 records. We see that, while the DASD processing time is quite sensitive to small and medium hit rates, the performance of CAFS is rather insensitive to it over the entire range. This is because a CAFS cell is much larger than a block in conventional DASD, and most cells would have to be searched anyway even for a relatively small record hit rate. Therefore, CAFS indexed sequential files in general do not benefit from skip sequential processing. Fig. 6 shows the processing time ratios of DASD to CAFS for the same file size with different hit rates. We see that this ratio increases rather sharply for small hit rates but gradually settles to about 48 for large hit rates. Experiments have also been performed for file sizes 5000, 10000, 15000, 20000, and 25000, and they are observed to exhibit a similar extent of improvement.

Additional experiments have been performed on CAFS with small cell sizes of 1 and 2 tracks, and we find that, somewhat surprisingly, that for a hit rate of 1%, CAFS actually exhibits a slight degradation in performance in comparison with conventional DASD: CAFS is slower by 28.7% for a cell size of 1, and by 0.4% for a cell size of 2. We also find that when the cell size is increased to 20, a slight degradation (about 4%) in performance is also evident in comparison with a 10 track cell size.

4.3 Evaluation of Hashed Random Organisation

In ICL hashed random files implemented in conventional DASD, buckets are accessed by means of a bucket directory comprising a fixed number of pointers, one for each bucket, each pointer being the address of the first block for that bucket. In any bucket, each block within it contains a pointer to the next block

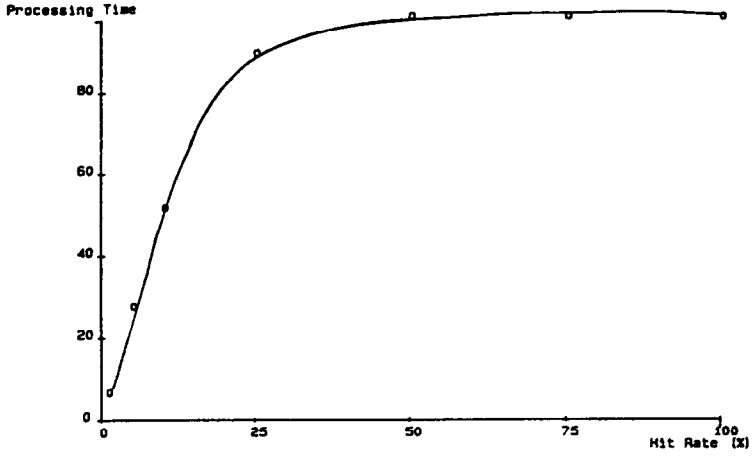

Fig. 4 DASD Processing Time (sec) - Skip Sequential Processing

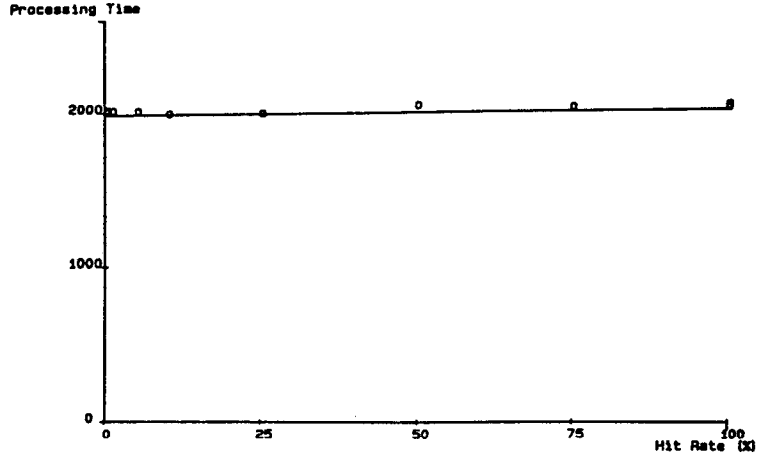

Fig. 5 CAFS Processing Time (ms) - Skip Sequential Processing

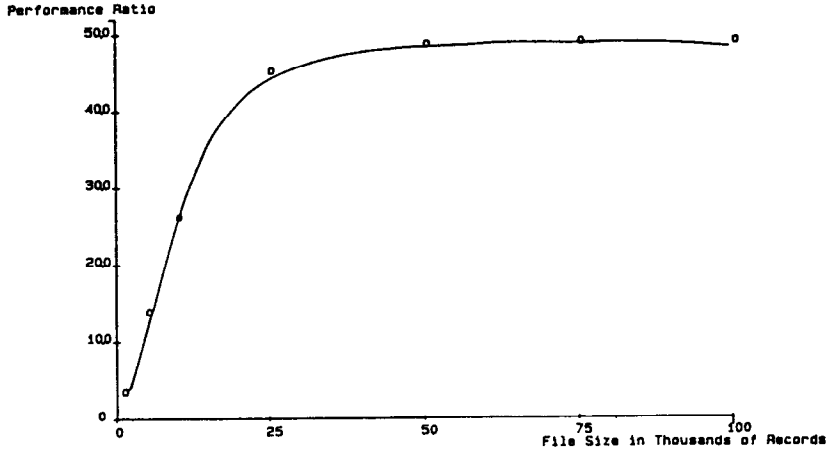

Fig. 6 Performance Ratio: DASD/CAFS - Skip Sequential Processing

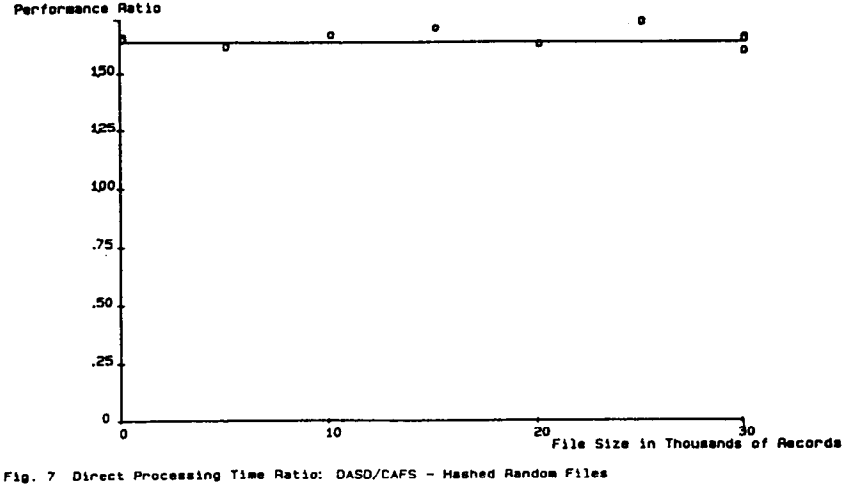

Fig. 7 Direct Processing Time Ratio: DASD/CAFS - Hashed Random Files

(if any) with the last (or only) block in that bucket containing a null pointer. Buckets are generally of variable size and consists of as many blocks as required. We assume that the whole bucket directory always resides in main memory when the file is being processed. Key-to-address transformation and searching of bucket directory and pointer manipulation are done in main memory and the time taken is assumed to be negligible. The space taken up by the pointer in each block is assumed to be insignificant. The number of blocks in a bucket depends on the number of records that have been hashed to this bucket address and the blocking factor. The probability p(x) of an address having x records allocated it it is assumed to conform to the Poisson distribution [2]:

$$p(x) = \frac{[(n/N)^x] \exp(-n/N)}{x!}, \quad x=0,1,2,...$$

where n signifies the number of records, and N signifies the number of buckets. The probability that there are x records in a bucket is taken to be $p(x)/[1-p(0)]$, which is used to decide the length of the search through a bucket. With a packing density of 90% and single record block, it was found that the bucket size would not exceed 9 blocks. It is therefore not unreasonable to assume that the size of each bucket does not exceed 10 blocks. These probabilities for the first ten blocks are calculated and stored in an array in the course of the simulation. Each record is accessed by computing the bucket number and consulting the bucket directory to locate the first block of that bucket. These are assumed to take negligible time. Each block of the bucket is then searched successively for the record. The time to search each block includes a random seek to the required cylinder, half a rotation for the block to come under the read/write heads, block transfer time and record processing time. At each block, the probability above is used to determine whether the record is found. Searching continues until the record is found or the maximum number of blocks have been searched. It is assumed that on average the required record can be found after searching half of the records in the block where it is stored. All records in blocks preceding the one where it is found would have to be searched.

With CAFS implementation, the situation is very similar except that since the smallest addressable unit is a cell, each bucket is assumed to be made up of a variable number of cells. The number of records each cell can hold is thus quite large. It was found that for a single track cell, the bucket size rarely exceeds two cells. As the cell size increases, the number of bucket addresses decreases and the ratio (n/N) becomes very large so that p(0) = exp(-n/N) tends to zero. It was found that for a cell holding more than 70 records, p(0) becomes so small that it cannot be represented even using a double precision number on the PDP 11/44 machine on which the simulation model is implemented. For this reason we simply assume that for a cell that can hold more than 70 records, the length of each bucket is less than or equal to 1 cell. This assumption is reasonable because for a packing density of 90% and a bucket size of 75 the percentage of synonyms as a percentage of records loaded is shown in [2] to be 1.25%. We assume that each cell access in a bucket requires a random seek, one block rotation delay and scanning through the cell. The position where the record is found is random and searching can stop as soon as it is found. If the record is not found in a cell, the whole cell would have been scanned before searching the next cell.

Fig. 7 displays the processing time ratios of DASD to CAFS for different file sizes under direct processing. We find that this ratio is well below 2; the intercept of the least square line through these points is calculated to be 1.64. This suggests that the improvement in efficiency in the direct processing of CAFS hashed random files is rather marginal. However, the experimental results relating to exhaustive searching of these files yield an average (least square) ratio of 45.5.

Experiments have also been performed to gauge the effect of the cell size on performance on a file of 32000 records and cell sizes of 2, 4, 5, 10, and 20; the same phenomenon observed in Section 4.2.1 is also present with an increase in processing time of about 23% being observed for a cell size of 20 tracks.

5. SUMMARY AND CONCLUSIONS

We have studied the performance advantages of CAFS in relation to conventional DASD through performing over 100 detailed simulation experiments. We find that the benefits conferred by CAFS vary quite considerably depending crucially on the file structure and the processing operation. In the case of sequentially organised files, conventional DASD is slower than CAFS by a factor of about 2.5 for direct processing under binary search which increases to about 50 for exhaustive search. In the case of indexed sequential files with external index searching, a comparable extent of improvement is observed. If, however, external index search in conventional DASD is avoided, then we find that CAFS fails to offer any tangible performance improvement for direct processing. We also find that, although indexed sequential files implemented on conventional DASD generally benefit from skip

sequential processing, the same fails to be true for CAFS implementations. We find that the latter implementation is rather insensitive to the hit rate; even with a relatively small hit rate of about 10%, a complete scan of the file frequently results. In fact, for a hit rate of 1%, CAFS could actually be slower than conventional DASD. Similarly, in the case of direct processing in a hashed random file, the performance advantage of CAFS is rather marginal: conventional DASD being slower than it on average by a factor of only 1.6. Exhaustive searching of a hashed random file with pre-specified selection criteria, however, does substantially raise this factor to about 50 times. These improvement factors appear to be quite general and is not affected by the file size, provided the file contains a minimum of several thousand records.

We also find that CAFS could be counter-productive if the wrong parameters are chosen for its implementation. These experiments suggest that for a system with drive amplification of degree n, then having a cell size which substantially deviates from n is inefficient, and the system normally behaves quite satisfactorily when the sucket size is the same as the degree of drive amplification.

Thus, in a processing environment where there is contention for CAFS resources, we would recommend that CAFS not be used for (i) direct processing of indexed sequential files with index loadable into main memory, (ii) skip sequential processing with expected hit rates of less than 1%, (iii) direct processing of hashed random files. Further tasks which might be off-loaded from CAFS without sacrificing too much efficiency are (i) direct processing of sequentially organised files, and (ii) direct processing of indexed sequential files with external index search.

The chief strength of CAFS is in exhaustive search. For highly complex search expressions, conventional DASD could be slower than CAFS by as much as a factor of 100; for moderately complex expressions, this factor is generally about 50. The merit of CAFS appears to be based on a rather simple design: the elimination of rotational delay by dedicated hardware which obviates time-consuming intermediate I/O transfers and memory references. This elimination takes 3 forms: (i) by allowing the fields to be self-identifying, any initial search delay caused by identification by position may be eliminated; (ii) by speeding up the necessary ordercode execution so that search may take place continuously without losing a rotation for each inter-block gap; (iii) by supporting the simultaneous reading of multiple tracks, only a single rotation is necessary for search tasks which otherwise may require several. As we have already indicated, since certain tasks are unsuited to CAFS, a highly commendable design feature of it is the inclusion of the Direct Access Unit, which conveniently allows tasks to be switched between CAFS mode and conventional mode. A key weakness of CAFS, however, is that it makes no attempt to reduce the seek time which accounts for a substantial proportion of the delay in many processing situations. This could be overcome by using a multi-head or fixed-head disc; a further improvement along this line would be to introduce an additional degree of parallelism by allowing several of these heads on different cylinders to be active simultaneously - theoretical work on this type of architectural design is already available (see e.g. [1]). With such additional enhancements, it is very likely that file operations in which there is currently only marginal improvement may also yield impressive performance gains.

REFERENCES

[1] Calderbank, A.R., E.G. Coffman Jr., and L. Flatto. Optimum head separation in a disk system with two read/write heads. **J. ACM**, Vol.31, No. 4, pp. 826-838, 1984.

[2] Hanson, O. **Design of Computer Data Files.** Pitman, 1982.

[3] ICL. **CAFS 800 General Enquiry System.** 1982.

[4] ICL. **VME/B File Management.** 1983.

[5] Leung, C.H.C. and Q.H. Choo. The effect of fixed-length record implementation on file system response. **Acta Informatica** Vol.17, pp. 399-409, 1982.

[6] Leung, C.H.C. Analysis of secondary storage fragmentation. **IEEE Trans. Software Eng.**, Vol. SE-9, pp.87-93, 1983.

[7] Maller, V.A.J. Information retrieval using the content addressable filestore. In Proc. **Information Processing 80**, North-Holland, pp. 187-192, 1980.

[8] Teorey, T.J. and J.P. Fry, **Design of Database Structures.** Prentice-Hall, 1982.